Dependability Considerations in Distributed Control Systems*

K. Žagar†, Cosylab, Ljubljana, Slovenia

Abstract

Metaphorically speaking, distributed control system is a messenger that carries orders from the 'commander' (the operator) to the 'battlefield' (devices), and reports the other way around. As such, it is of central importance for the functioning of a large experimental physics facility.

The exchanged messages must be delivered reliably: once sent, the message should reach its destination. The messenger should be highly available: whenever a message is to be sent, the sender shall not wait to be able to do so. Last but not least, the transmission should be secure: the messages should not be tampered with while en-route. In short: the control system must be dependable.

In this article, ongoing research in the field of dependable distributed systems performed in the context of the 6th European Framework Programme's DeDiSys project is presented. First, dependability issues of distributed control systems are analyzed. Then, guidelines are given on how, where and why to use techniques that improve dependability. In particular, the availability aspect of control systems is discussed, and potential for improving availability by trading it against consistency is investigated. Finally, dependability efforts we have invested in our products (in particular the ACS control system infrastructure) are described.

INTRODUCTION

Humans are increasingly more dependent on technology. However, such dependency may back-fire, hurting technology users in the long run.

Property of a technological solution that it may be depended upon under a wide range of circumstances is called *dependability*. Dependability involves the following concepts:

- **Reliability:** the solution must perform a task it has been designed for whenever it is requested to do so.
- **Availability:** the solution must be available all the time for which it has been designed.
- **Security:** malicious third parties must not be able to take control of the solution and use it against or without knowledge of its authorized users.

*The work described herein is supported by the European Community under Framework Programme 6 of the IST Programme within the project Dependable Distributed Systems (DeDiSys, Contract No. 004152)

† klemen.zagar@cosylab.com

This article is about dependability in distributed control systems. Topic of security in control systems has already been addressed at the previous ICALEPCS conference [2]. Therefore, this article will focus on the remaining two aspects.

Under normal conditions, control systems are usually reliable and available. However, unpredictable environmental influence, such as hardware failures and broken network links, may affect both reliability and availability. The ability of a system to remain resilient to such unpredictable events is called *fault tolerance.*

The DeDiSys Project

Cosylab is actively participating in the *Dependable Distributed Systems* (DeDiSys) project. The project is partially founded by the European Community under Framework Programme 6 of the IST Programme. The project brings together four industrial and four academic organisations from five European countries.

The goal of the project is to investigate whether a distributed information system can be built whose availability is increased in exchange for reduction of data consistency that the system is processing. Of course, long-term inconsistency of data would render the system useless, so a capability must be built-in to reconcile any inconsistencies in data that have occured while the system was in a degraded mode of operation.

During the project, several approaches to achieve this availability/consistency trade-off will be attempted by building prototypes. Once the issues are understood, Cosylab may use the acquired knowledge and prototypes to improve the distributed control system infrastructures, in particular ACS and EPICS [3].

FAULT TOLERANCE IN DISTRIBUTED SYSTEMS

In distributed systems data and services are located on more than one node (computer), and the nodes are interconnected using a network so that they may gain access to each other's resources.

If no additional measures are taken, this implies that many or all of the computers in a distributed system must be fault-free in order for the system to be available. For example, if a business application uses a client computer to present the user interface to the user, a server to handle the business logic, and a database to manage persistent store of the data, all three computers must be functional in order for a typical operation to be able to commence. The probability of at least one computer out of three failing is approximately three times greater as the probability of a single computer's failure. Consequently, such a distributed system would be approximately three times less reliable (and thus also less available) than a single-computer solution. All three components in this case ' the client, the server and the database ' are *single-points-of-failure.*

To counter this effect, data and services are replicated. By replicating a resource on two computers, both computers must fail in order for the resource to be rendered unavailable. The probability of two computers failing, however, is significantly smaller than the probability of a single computer's failure. For example, if a single computer has a 10% chance of failing in, say, a week, two computers would both fail only with probability of 1% (10% of 10%), which is an order-of-magnitude improvement.

But replicas bring about complexities of their own. One of the most important issues is how to implement a mechanism that handles replication. If a replica on one host is modified, this update should propagate to all the other replicas. The propagation could be either *synchronous* (all other replicas are updated before accepting the modification of the local replica), or *asynchronous* (the modification is accepted whereas the other replicas are updated at some later time). The synchronous case ensures that all the replicas are consistent, whereas in the asynchronous case the replicas may become temporarily inconsistent. On the other hand, the synchronous case does not solve the single-point-of-failure problem described above: as soon as a host of one of the replicas is unavailable, the synchronous propagation of replica's state can not commence.

A prototype system will be built that will strive to demonstrate practical benefits of improving availability at the cost of consistency. If the prototype is successful, existing industrial control systems such as ACS would be adjusted to take advantage of the concept. Ultimately, an industrial control system that features configurable availability/consistency trade-off would allow for finding a point close to a cost optimum of a plant or facility where such a control system will be deployed.

FAULT TOLERANCE IN ACS

The particular technology that served as the basis of our investigation is ACS. ACS (*Advanced Control System/Atacama Large Millimeter Array Common Software*) is an open-source component-oriented infrastructure for building distributed control systems.

Components representing controlled devices or control logic can be deployed across host computers throughout the network. A central entity called the Manager is responsible for determining on which host a given component will reside. This centralized approach to deployment allows dynamic reconfiguration of the system, e.g., due to changes in requirements or as an automated response to failures within the system.

Apart from managing the lifecycle of components, ACS also provides other infrastructural services, such as distributed logging, fault-detection, centralized configuration database, load-balancing and asynchronous publisher-subscriber communication mechanisms.

ACS is built atop of CORBA middleware. ACS components and clients can be written in several programming languages (C++, Java or Python) and can be hosted by a variety of operating

systems (many flavours of Linux, Solaris and Windows).

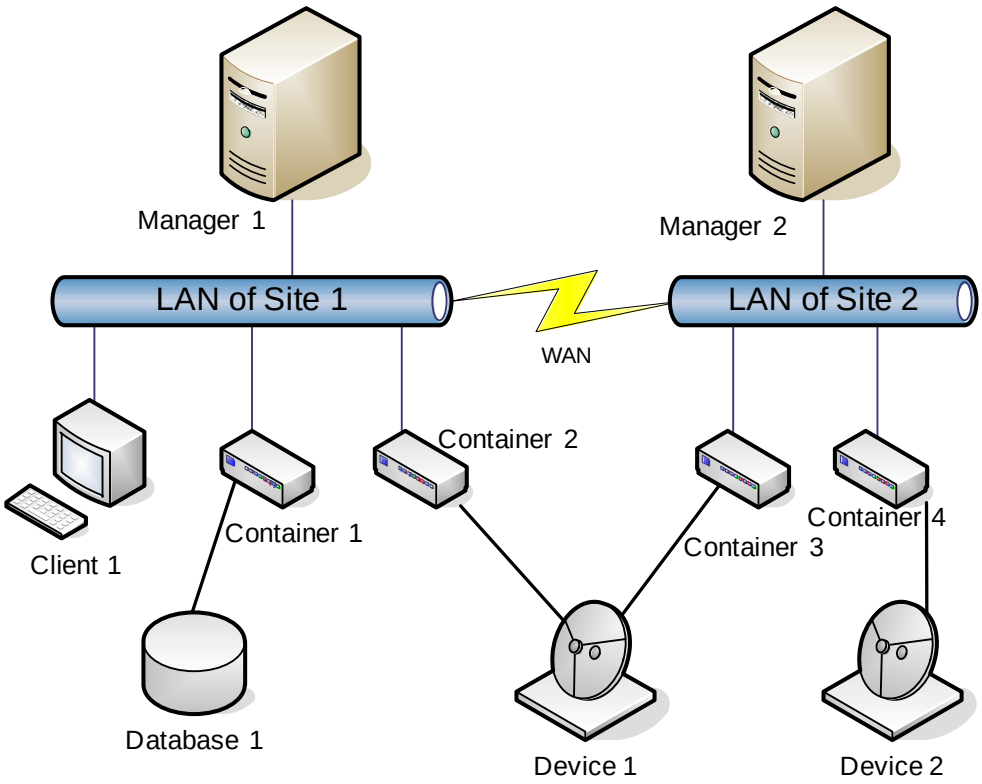

Figure 1: an example ACS deployment. Two sites are connected via a Wide Area Network (WAN) link. Containers are capable of hosting components that provide application-specific logic, in this particular case, drivers for hardware devices and database access. The client may access any component in the system through the manager, which supervises and controls the deployment.

In ACS, like in any other distributed system, the individual nodes and their links are expected to fail. Depending on the criticality of the failed node, the ACS may be unable to continue with normal operations. In some cases, temporary disruption of a node has caused other parts of the system to malfunction as well. This avalanche effect is mostly due to improper development practices of the application developers, who fail to account for all the possible conditions that may occur in a distributed system. However, these issues should not be of application developer's concern – the ACS should provide reasonable fault-tolerant behaviour on the application's behalf.

Applications

Currently, the primary use of ACS in large experimental physics controls. In Karlsruhe (Germany), ACS is used for control of an ANKA synchrotron light source [4]. ACS will also be the

underlying infrastructure for the *Atacama Large Millimeter Array* (ALMA, being built in co-operation of the *European Southern Observatory* and the *North American Radio Observatory*), which will be the largest array of radio-telescopes currently under construction in the Atacama Desert [5][6]. Further applications of ACS include the controls for the 1.5 meter *Hexapod Telescope* (*Ruhr University Bochum*), the *Atacama Pathfinder Experiment* radio-astronomy antenna (APEX, *Max Planck Institute for Radio Astronomy*), the 40 meter radio-antenna of the *Observatorio astronomico national* [7] and the 64 meter *Sardinia Radio Telescope* [8].

The usability of ACS transcends the boundaries of distributed control systems. In its essence, ACS is an infrastructure for component-oriented distributed systems, making it suitable for use also in classical business scenarios. For example, Cosylab has successfully applied load-balancing features of ACS to build a scalable Geographical Information System.

Failure scenarios in ACS

The following failure scenarios have been identified for the case of an ACS-based distributed control system:

1. **WAN link failure.** The link between two sites fails, essentially resulting in a **network split**. None of the subnets becomes entirely unavailable, as all the crucial services (e.g., the manager) are available in all of them. However, the subnets may evolve independently of each other, resulting in possible difficulties when they are re-joined (e.g., each of the subnets bringing up a service whose state must then be reconciled).

2. **LAN failure.** LAN failure may come in two forms: either link to a single host is lost, or the entire LAN infrastructure (e.g., a router or a hub) fails. The isolated node (or nodes) must decide what actions to take so as not to interfere with those of the rest of the system. For example, the node could decide to continue execution of its services (e.g., closed-loop control), or terminate. Similarly, the node should assume a reasonable line of action if it boots in an environment without network connectivity.

3. **Node crash.** A node crashes unexpectedly (e.g., a RAM or CPU failure). Unfortunately, to the rest of the system this kind of failure is difficult to differentiate from connectivity failure to the crashed node.

4. **Self-discovered fault of a node.** The node discovers a fault and terminates gracefully. During termination, it lets the rest of the system know of its condition.

5. **Manager unavailability.** Currently, the manager is a single-point-of-failure, as it can not be replicated yet. This shortcoming is not extremely critical, because the manager is capable of recovering its state upon restart, and its availability is a necessity only during

start-up or reconfiguration of the distributed control system. However, to achieve higher availability rates for the entire system, the manager itself should also be replicated.

All of these scenarios should be incorporated into middleware or a library, whose application programming interface (API) would be easy to use, but would still allow a lot of flexibility (for example, vetoing and application-level assistance during reconciliation of network splits).

CONCLUSIONS

The number of control points and the complexity of their interactions in control systems are increasing. To maintain a fair level of control, the resulting distributed control systems must remain dependable in spite of this increased complexity. To improve dependability, effects of faults must be prevented from destabilizing the entire system, and possibly mitigated automatically to amend minor issues. Before attempting to resolve this problem, the nature of faults must first be thoroughly understood. Therefore, the description of various fault scenarios, their classification, and recommendations for their mitigation are one major subject of this article. Apart from this, the article discusses the potential for improving availability by risking constraint consistency of the system's constituents. Finally, it proposes amendments to existing middleware that would make the benefits of configurable availability/consistency trade-off available to controls developers without requiring much of their effort.

REFERENCES

[1] Cosylab, http://www.cosylab.com

[2] K. Žagar: "Security Considerations of Distributed Control Systems", ICALEPCS 2003, Gyeongju, Korea, October 2003

[3] Experimental Physics and Industrial Control System (EPICS): `http://www.aps.anl.gov/epics`

[4] I. Križnar, W. Mexner et al: "The Upgrade of the ANKA Control System to ACS (Advanced Control System)", ICALEPCS 2003, Gyeongju, Korea, October 2003

[5] G. Chiozzi, K. Žagar et al: "The ALMA Common Software (ACS): Status and Developments", ICALEPCS 2003, Gyeongju, Korea, October 2003

[6] Atacama Large Millimeter Array (ALMA): `http://alma.nrao.edu`

[7] de Vicente, P. Bolaño: "Software tools and preliminary design of a control system for the 40m OAN radiotelescope", Proceedings of the Astronomical Data Analysis Software and Systems 2003 Conference, Strasbourg, France

[8] Sardinia Radio Telescope: `http://www.ca.astro.it/srt`